d-tree V3.1 Release E4.2

Update Notes

The following sections describe modifications and additions to d-tree since release E3, not yet noted in the documentation. Steady headway is being made on new documentaion. This new manual will be sent to all licensed users as soon as possible.

1) Installation Notes: This release is distributed on 8 (360kb DOS) diskettes. Disks 1 thru 5 contain the d-tree source code. Disk 6 contains subdirectories with "make" files as described in the d-tree manual. Disk 7 contains files used by the "group" tutorial. This tutorial is a new addition in this release (see note 3 below). Disk 8 contains a custom editor used in the tutorials (used in DOS only).

- **DOS Users:** Follow the normal installation as described in the manual, except note that you will be prompted for 8 diskettes. **Turbo C users:** check the batch files used to compile in the "turboc" directory. You may need to place a path name in front of the turboc objects and libraries (example: where you see "c0l.obj" may need to be "\tc\lib\c0l.obj). **Microsoft users:** set your INCLUDE environment variable to the d-tree directory.
- **Xenix Users:** follow the normal installation utilizing the additional DOSMOVE files. You now have DOSMOVE1 thru DOSMOVE7. Disk 8 only applies to the DOS environment.
- **UNIX Users:** Copy disks 1 thru 5 to your d-tree directory on your Unix box (i.e.: /usr/dtree). Copy the Unix subdirectory on disk 6 to this same directory. On your Unix box, create a subdirectory within your d-tree directory named "tutorial" (i.e.: /usr/dtree/tutorial). Copy disk 7 to this "tutorial" directory. This directory must be spelled "tutorial", and be a subdirectory of your primary d-tree directory for the group tutorial to work. There is a file called "dtshoot.txt" on disk 6 within the Unix directory which contains all the file names. This may help if you are using a communications utility to port the source code. (Special Note: The "group" tutorial does not handle running in c-tree's "unifrmat" mode).

2) Easier to follow version of DT_SCORE.C: We understand that many users are following our "dt_score.c" mainline as the base for their applications. d-tree uses this mainline in multiple ways. The "run" program is based on this mainline. When a maintenance program is created from the catalog, it utilizes "dt_score.c". It is also the mainline used when a "HARD CODED" version of the program is generated. Because of all these uses, "dt_score.c" (with all its #define's) is not the easiest logic to study as an example for your applications. We are evolving a new version of "dt_score.c" and have named it "**dt_smore.c**".

We have left "dt_score.c" as is, so users who have used this as a base still have the old version. We now suggest using "dt_smore" as your guide. During our effort to simplify our mainline, some new features have evolved:

- **New d-tree function calls to handle ADD, CHANGE, DELETE, and MAINTAIN records:** The first step in simplifying our mainline was to modulate the add, change , delete, and maintain record routines. We have created d-tree function calls to handle this need. By doing this we not only were able to clean up our mainline, but provide functions that will make your programming easier. Note the functions DT_ADDMD, DT_CHGMD, DT_DELMD, and DT_DELMD in the source files of the same name. Also look at the "dt_smore.c" mainline to see them applied.
- **New IMAGE keywords - RELATED_SFL and RELATED_IMAGE:** One issue that had to be resolved when creating the new function calls mentioned above was how to handle subfile processing, without the need for a #define. (remember how we were using #define SFL in "dt_score.c" to indicate subfiles). To solve this problem, we added the new IMAGE keyword RELATED_SFL. Placing this keyword on a IMAGE definition, indicates that this subfile "relates" to this IMAGE. When this IMAGE is displayed and/or maintained, the subfile will also be displayed and/or maintained. The same applies to "RELATED_IMAGE" except you specify an IMAGE rather than a subfile. Note the following syntax:
  IMAGE(master) {LSTFLD_ADVANCE} {RELATED_SFL=trans}

3) Group Example/Hard Coded Pgm Illustration:

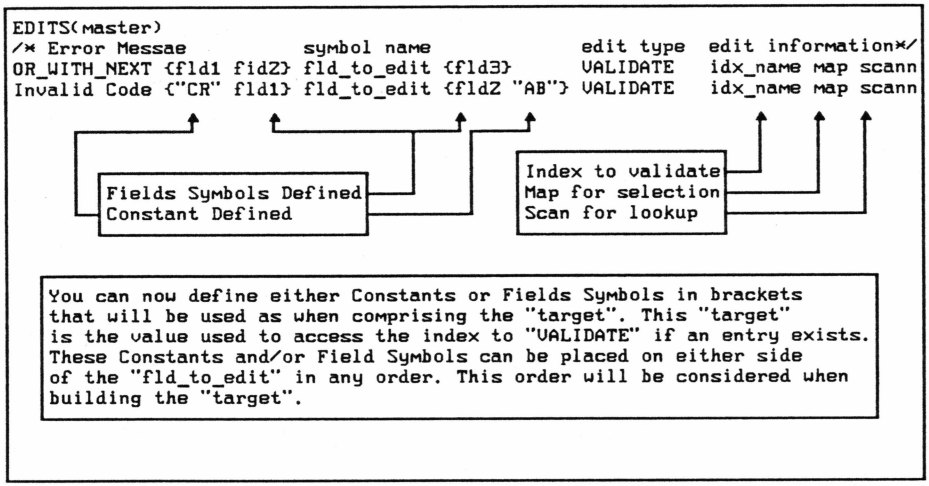
One of the most powerful capabilities found in d-tree, resides in a concept called "GROUPS". When used in conjunction with the DATA_DICTIONARY ability, "GROUPS" allow you to create programs that "swap" ability and file definitions in and out of memory at runtime. This allows you to design generic mainlines, that are "Data and Application Independant". In order to illustrate this concept, we have put together an "automated tutorial". The "make" files distributed with this release compile a new program called "tutorial" . This "automated tutorial" picks up where the tutorial in the d-tree manual leaves off. The first option will initialize the data base and the d-tree script to the state as if you had just finished tutorial sessions 1 thru 4 in the documentation.

- Users who have any problems with these sessions can do the following. Run this new program by simply keying in "tutorial". Select the first option to initialize the data and scripts. Return back to your operating system and look at the scripts and the catalog. Looking at the finished version should help.

Run the first six (6) options in the order shown on the menu. Once completed you will have a single executable, maintaining five different file and screen groups. Look at the "dt_smore.c" mainline to understand the logic.

The last option on the tutorial menu (option A.) will illustrate the "-c" switch used in the dt_score.c and dt_smore.c mainlines. This is the switch which will create a "HARD CODED" version of a program, by calling the DT_COMPL function. This function converts the definitions contained in a d-tree script into 'C' source code. This source coded is then compiled and linked with the mainline to produce a stand alone "hard coded" program. Note: Running this option will set the "customer.dts" script as it appeared in session 1 of the tutorial in the manual. Don't let this confuse you when looking at the scripts.

4) VALIDATE Edit Enhanced: Mutiple key segments support has been added to the VALIDATE edit type in the EDITS ability. Placing either a field symbol name or a constant value within brackets will indicate to d-tree to consider these values when comprising the key value used in the equal access operation. Because these field or constant definitions can reside both before and/or after the primary field, this enhancement provides a means to define both "prefix" as well as "suffix" values to use in the index target. See illustration below:



5) MANDATORY_IF Enhanced: The MANDATORY_IF edit type in the EDITS ability now allows more than one field to be considered when making the determination if the MANDATORY edit should be performed. In otherwords, you can define a field to be MANDATORY only if all the fields defined within brackets have data. See illustration below:

```
EDITS(master)
/* Error Message          symbol name   edit type     edit information */
Must Enter This Field     fld_to_edit   MANDATORY_IF {fld1 fld2 fld3}

        In this example the "fld_to_edit" will only be checked
        for MANDATORY entry if all fields within the brackets
        have data (ie: if fld1, fld2, and fld3 all have data).
```

6) New IMAGE keywords: {MESSAGE_LINE} {DATA_ONLY} {RELATED_SFL} {RELATED_IMAGE}:

Four new keywords have been added to the IMAGE ability. Placing the **MESSAGE_LINE** keyword as shown below will override the default message line while processing this screen. **DATA_ONLY** will cause the "image out" routine to only output the variable fields (not constant data...Titles, etc.) if this IMAGE was the last image in the image log (was the last screen displayed). This attribute is useful in an add mode, to prevent the entire screen from being redrawn after each add. As mentioned above RELATED_SFL and RELATED_IMAGE allow subfiles and other images to be connected (or related) to this IMAGE. When this image is displayed, all its related subfiles and images are also displayed. When this image accepts input, all related subfiles and images are also maintained.

```
IMAGE(refname)  {POP_UP} {RELATED_SFL=sfl_ref} {RELATED_IMAGE=image_ref}
                         {MESSAGE_LINE=8} {DATA_ONLY}
 +
   @DATE       FairCom Database          @TIME
           This is my example screen

            Enter Data Here: _______
            More Data Here:  ___________

            Message line will be here
                                                  +

FIELD(refname)
/* Symbol Name   Input Attribute        Output Attribute   Input Order */
   data1              NONE                   NONE               1
   data2              NONE                   NONE               2
```

This will cause the default message line to be changed to line 8 when processing this IMAGE.

This attribute will cause only the data fields to be displayed if this IMAGE is displayed more than one time consecutively.

7) More Input Attributes Allowed: The input attribute definition in the FIELDS definitions structure has been changed from an int to a long, thus allowing up to 32 input attributes per field.

8) New CONFIG Ability: A new ability has been added to allow the definition of global configuration information. The CONFIG ability allows the user to change the following information from within the script:

- **Default Colors:** Define the defaults foreground and background colors to be used when displaying IMAGEs. If not defined, screens will default to Yellow on Black unless otherwise defined by FORGROUND/BACKGROUND keywords. (See DT_TYPDF.H for where Yellow on Black is defined as default color).
- **Default Message Line:** Define an alternative message line. If not defined, the message line defaults to line 24. (see DT_TYPDF.H for where the 24 is defined).
- **Defaults Field Input Brackets:** Defines the Input Guides that are placed on either side of a field when the cursor travels into that field for input. Also define the characters that are to be displayed in those positions when the cursor moves out of that field. Defaults to [ ] when in the field and blanks upon exiting the field.

```
CONFIG(master)

COLORS         YELLOW BLACK /* Yellow foreground with Black background */
MESSAGE_LINE   24           /* this is the default message line */
INPUT_GUIDES   [ ]          /* brackets surround field when cursor is
                               in the field for input */
INPUT_SIDES    SPACE SPACE /* After cursor leaves field the brackets are
                               replaced with spaces */

            The above configuration presents the default values
            when no CONFIG keyword is defined. Use the CONFIG
            keyword in your script when you want to change any one
            or all of these default values. See the following example:

CONFIG(master)
COLORS         WHITE BLUE   /* Changes default IMAGE colors to WHITE on BLUE */
MESSAGE_LINE   25           /* Change default message line */
INPUT_GUIDES   NONE NONE    /* define NONE so that no GUIDES will be used.
                               This is common when INPUTRI is defined as a
                               output attribute for the fields */
INPUT_SIDES    NONE NONE    /* nothing be outputted on either side of
                               field when cursor leaves field */
```

9) REVERSE IMAGE COLOR: The default for the "reverse image" attribute on a color monitor is to simply switch the foreground with the background. The user can define a specific color in TERMCAP to be used at all times for the RI attribute. The TERMCAP examples are shown below:

```
TERMINAL(DOSCOLOR)
RI    0          <-- This defines that reverse image will
LOTUS 0          <-- reverse forground with background.


TERMINAL(DOSCOLOR)
RI    -31 -28    <-- This defines that reverse image will always be
LOTUS -31 -28    <-- white on red.

                 Note: Colors are define in DT_TYPDF.H
                 The following is a snapshot of part of
                 DT_TYPDF.H.

#define DTSCRED    (-28) /* red                    */
#define DTSCMAG    (-29) /* magenta                */
#define DTSCBROWN  (-30) /* brown                  */
#define DTSCWHITE  (-31) /* white                  */
```

10) Keyboard Capture/Playback: A new function has been added to d-tree which will record keystrokes on disk and then allow them to be played back. This option is an excellent way to create demos for your applications written with d-tree. The function DT_CAPTR in DT_MISCI.C can be used to provoke the following:

- Start Keystroke Capture.
- Stop Keystroke Capture.
- Playback Keystrokes.

We have implemented this function call in the dt_score.c mainline in the following manner. We will use the "RUN" program as an example. Execute the program with the "-t" (trap) switch to turn on the keystroke trapping like so:

C> run -t customer.dts

Every key hit will be stored in a file called "strokes". When you escape out of "run" you will then have stored all keystrokes during that session. To play the keystrokes back use the "-d" (demo) switch to indicate to "RUN" which keystroke file to use. First, save this sessions strokes as so:

C> copy strokes demo1

Now run your demo (remember: if your session added records, you may want to delete the data and index files before playing back the keystrokes):

C> run -ddemo1 customer.dts

The "playback" mode simply redirects the getchar logic to get the characters from the "strokes" file. See DT_MISCI.C for DT_CAPTR() function.

11) Low Level Pad and Un-Pad: The two new functions DT_LUPAD (low level unpad) and DT_LOPAD (low level pad) were added to allow addressing a particular record buffer. Note: DT_DOPAD and DT_UNPAD assume record buff-40H er[0]. These new functions allow the record buffer to be sent as a parameter. See DT_CTREE.C for code.

12) General Parsing Program: We have added a general parsing program that can be used to create either "group" files for hard coded specs. The program DT_GPARS (d-tree general parse) will parse d-tree scripts. If a GROUP keyword exists in the script, the definitions will be written to the defined "group" file. The script name is passed as the first parameter to the DT_GPARS program. An optional second parameter can be passed. This parameter indicates the file name to write hard coded specs. Note the following example will parse the "customer.dts" script and write its definition to the "customer.h" file.

C> dt_gpars customer.dts customer.h

13) Ability Structure Sizes Changed: We have added some fields to the IMAGE, FIELD and SCANN structures in DT_TYPDF.H. This means that if you are using groups, you will need to re-create your group files. We have provided a #define to allow these changes to be left out which will allow your old group files to work. See #define DT31E2 in DT_TYPDF.H. Leaving these new fields out will deactivate the MESSAGE_LINE and 32 Input attributes described above.

14) SUBFILE RECORD EDIT: We have added a call to DT_EDITS (the edit function) for each subfile record as it is entered. Example: This will catch things like MANDATORY fields when the user hits the down key to advance to the next subfile record. For users who do not want to perform this edit on each subfile record, we have added a new subfile attribute. Placing the attribute "**NO_EDIT_RCD**" within the SFL_ATTRIBUTE section in a subfile definition will relax this edit.

BUG FIXES

We would like to express our special thanks to all the users who have taken the time to report problems. The following problems have been resolved:

- Record structure alignment (6802 errors) have been resolved.
- Reread edit in multiuser mode not freeing lock if a write failed.
- Reread edit was causing a c-tree error 4 on c-tree's rewrite operation.
- Error when defaulting one MONEY field type to another MONEY type when using the DEFAULTS ability.
- DFALT_LIST type DEFAULT was not handling one byte char fields correctly.
- Underlines that display when a field had no data was inconsistent. Sometimes a field had no data and the underlines would not display on the screen.
- The catalog program defaults key segment modes based on field type. If the user had keyed a desired segment mode of 3 or 12, the catalog was changing it.
- The logic which built "target" fields used to load subfiles was assuming a null byte. The logic has been changed to consider the segment definitions for the key and no special assumptions are made.
- When exiting a date field with an up key (or anything other that a CR), garbage was being placed in the date field.
- There was a bug in the MAP ability parsing routine when a length was defined.
- If a field was defined as NUMERIC and the field was full (there is a character in each postition), the field would be blanked out.
- There was a bug when numbers were entered with a lot of decimal places. A value of 0.00009 was displaying as _____._____.
- Underline and reverse image attributes were not being preserved when characters were being entered.
- When doing a "system" call from a menu, the screen was not being redisplayed correctly when control returned to the menu program.
- We had numerous problems in our color support. Reverse image, jagged edges, INPUTRI problems have been fixed.
- We have fixed the MONEY type rounding problem when negative numbers are entered.
- The IMAGE parsing routine was incorrectly detecting the rightmost position of a screen.

- There was a problem with the DUPKEY edit when the field was in a subfile. This was causing a "Index name must be unique" in the catalog when reviewing index definitions.
- Subfile functions have been cleaned up. The equal subfile function was always returning the first subfile record.
- There were problems when refreshing screens involving POP-UPs.
- The DT_DDICT ability when defining multiple IFILS was not setting the ix and segs pointers correctly in the IFILS and IIDX structures.
- Catalog select files option problems resolved.
- Subfile redisplay from image log problems resolved.

We cleaned up a lot of other miscellaneous problems. If you are still having a problem after trying it with this release, please let us know. Thanks again for your support.

END UPDATE NOTES